16-385

Method and Apparatus for Using Conditional Selectivity as Foundation for Exploiting Statistics on Query Expressions

Technical Field

The invention relates generally to the field of relational databases and specifically to the field of optimizing queries on databases.

Background of the Invention

Most query optimizers for relational database management systems (RDBMS) rely on a cost model to choose the best possible query execution plan for a given query. Thus, the quality of the query execution plan depends on the accuracy of cost estimates. Cost estimates, in turn, crucially depend on cardinality estimations of various sub-plans (intermediate results) generated during optimization. Traditionally, query optimizers use statistics built over base tables for cardinality estimates, and assume independence while propagating these base-table statistics through the query plans. However, it is widely recognized that such cardinality estimates can be off by orders of magnitude. Therefore, the traditional propagation of statistics can lead the query optimizer to choose significantly low-quality execution plans.

Summary of the Invention

Using conditional selectivity as a framework for manipulating query plans to leverage statistical information on intermediate query results can result in more efficient query plans. A number of tuples returned by a database query having a set of predicates that each reference a set of database tables can be approximated. The query is decomposed to form a product of partial conditional selectivity expressions. The partial conditional selectivity expressions are then matched with stored statistics on query expressions to obtain estimated partial conditional selectivity values. The selectivity of the query is then estimated by combining the obtained partial conditional selectivity results. The resulting query selectivity estimate can be multiplied by a Cartesian product of the tables referenced in the query to arrive at a cardinality value.

The decomposition of the query can be performed recursively by repeatedly separating conditional selectivity expressions into atomic decompositions. During matching an error can be associated with a selectivity estimation that is generated using a given statistic and those statistics with the lowest error may be selected to generate the query selectivity estimation. The error may be based on the difference between a statistic that is generated by an intermediate query result and a statistic on the corresponding base table. Statistics on query expressions that correspond to a subset of the predicates represented in a given selectivity expression may be considered for estimating the selectivity of the given selectivity expression. In an optimizer environment, the decomposition may be guided by the sub-plans generated by the optimizer. A wider variety of queries can be decomposed by transforming disjunctive query predicates into conjunctive query predicates.

Brief Description of the Drawings

The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings, in which:

Figure 1 illustrates an exemplary operating environment for a system for evaluating database queries using statistics maintained on intermediate query results;

Figure 2 is a block diagram of a prior art optimizer that can be used in conjunction with the present invention;

Figure 3 is tree diagram for a query and two alternative execution sub-plans for a prior art optimizer;

Figure 4 is a block diagram for a method for evaluating database queries using statistics maintained on intermediate query results according to an embodiment of the present invention;

Figure 5 is block diagram of a memo table for an optimizer that implements the method of Figure 4; and

Figure 6 is a tree diagram that illustrates a coalescing grouping transformation of query that can be used in practice of an embodiment of the present invention; and

Figure 7 is a tree diagram that illustrates an invariant grouping transformation of a query that can be used in practice of an embodiment of the present invention.

Detailed Description of the Preferred Embodiment

Exemplary Operating Environment

Figure 1 and the following discussion are intended to provide a brief, general description of a suitable computing environment in which the invention may be implemented. Although not required, the invention will be described in the general context of computer-executable instructions, such as program modules, being executed by a personal computer. Generally, program modules include routines, programs, objects, components, data structures, etc., that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

With reference to Figure 1, an exemplary system for implementing the invention includes a general purpose computing device in the form of a conventional personal computer 20, including a processing unit 21, a system memory 22, and a system bus 24 that couples various system components including system memory 22 to processing unit 21. System bus 23 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. System memory 22 includes read only memory (ROM) 24 and random access memory (RAM) 25. A basic input/output system (BIOS) 26, containing the basic routines that help to transfer information between elements within personal computer 20, such as during start-up, is stored in ROM 24. Personal computer 20 further includes a hard disk drive 27 for reading from and writing to a hard disk, a magnetic disk drive 28 for reading from or writing to a removable magnetic disk 29 and an optical disk drive 30 for reading from or writing to a removable optical disk 31 such as a CD ROM or other optical media. Hard disk drive 27, magnetic disk drive 28, and optical disk drive 30 are connected to system bus 23 by a hard disk drive interface 32, a magnetic disk drive interface 33, and an optical drive interface 34, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer-readable instructions, data structures, program modules and other data for personal computer 20. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 29 and a removable optical disk 31, it should be appreciated by those skilled in the art that other types of computer-readable media which can store data that is accessible by computer, such as random access memories (RAMs), read only memories (ROMs), and the like may also be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk, magnetic disk 129, optical disk 31, ROM 24 or RAM 25, including an operating system 35, one or more application programs 36, other program modules 37, and program data 38. A database system 55 may also be stored on the hard disk, magnetic disk 29, optical disk 31, ROM 24 or RAM 25. A user may enter commands and information into personal computer 20 through input devices such as a keyboard 40 and pointing device 42. Other input devices may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to processing unit 21 through a serial port interface 46 that is coupled to system bus 23, but may be connected by other interfaces, such as a parallel port, game port or a universal serial bus (USB). A monitor 47 or other type of display device is also connected to system bus 23 via an interface, such as a video adapter 48. In addition to the monitor, personal computers typically include other peripheral output devices such as speakers and printers.

Personal computer 20 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 49. Remote computer 49 may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to personal computer 20, although only a memory storage device 50 has been illustrated in Figure 1. The logical connections depicted in Figure 1 include local area network (LAN) 51 and a wide area network (WAN) 52. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

When using a LAN networking environment, personal computer 20 is connected to local network 51 through a network interface or adapter 53. When used in a WAN networking environment, personal computer 20 typically includes a modem 54 or other means for establishing communication over wide area network 52, such as the Internet. Modem 54, which may be internal or external, is connected to system bus 23 via serial port interface 46. In a networked environment, program modules depicted relative to personal computer 20, or portions thereof, may be stored in remote memory storage device 50. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Cost Estimation Using Cardinality Estimates Based on Statistics on Intermediate Tables

SITs are statistics built over the results of query expressions or intermediate tables, and their purpose is to eliminate error propagation through query plan operators. For the purposes of this description, a SIT is defined as follows: Let *R* be a table, *A* an attribute of R, and Q an SQL query that contains $R.A$ in the SELECT clause. SIT($R.A|Q$) is the statistic for attribute A on the result of the executing query expression Q. Q is called the generating query expression of SIT ($R.A|Q$). This definition can be extended for multi-attribute statistics. Furthermore, the definition can be used as the basis for extending the CREATE STATISTICS statement in SQL where instead of specifying the table name of the query, more general query expression such as a table valued expression can be used.

In U.S. Patent Application Serial No. 10/191,822, incorporated herein by reference in its entirety, the concept of SITs was introduced. A particular method of adapting a prior art query optimizer to access and utilize a preexisting set of SITs for cost estimation was described in detail in this application, which method is summarized here briefly as background information.

Referring to Figure 2, the query optimizer examines an input query and generates a query execution plan that most efficiently returns the results sought by the query in terms of cost. The cost estimation module and its imbedded cardinality estimation module can be modified to utilize statistics on query expressions, or intermediate tables, to improve the accuracy of cardinality estimates.

In general, the use of SITs is enabled by implementing a wrapper (shown in phantom in Figure 2) on top of the original cardinality estimation module of the RDBMS. During the optimization of a single query, the wrapper will be called many times, once for each different query sub-plan enumerated by the optimizer. Each time the query optimizer invokes the modified cardinality estimation module with a query plan, this input plan is transformed by the wrapper into another one that exploits SITs. The cardinality estimation module uses the input plan to arrive at a potentially more accurate cardinality estimation that is returned to the query optimizer. The transformed query plan is thus a temporary structure used by the modified cardinality and is not used for query execution.

According to the embodiment described in application serial number 10/191,822, the transformed plan that is passed to the cardinality estimation module exploits applicable SITs to enable a potentially more accurate cardinality estimate. The original cardinality estimation module requires little or no modification to accept the transformed plan as input. The transformation of plans is performed efficiently, which is important because the transformation will be used for several sub-plans for a single query optimization.

In general, there will be no SIT that matches a given plan exactly. Instead, several SITs might be used for to some (perhaps overlapping) portions of the input plan. The embodiment described in application serial number 10/191,822 integrates SITs with cardinality estimation routines by transforming the input plan into an equivalent one that exploits SITs as much as possible. The transformation step is based on a greedy procedure that selects which SITs to apply at each iteration, so that the number of independence assumptions during the estimation for the transformed query plan is minimized. Identifying whether or not a SIT is applicable to a given plan leverages materialized view matching techniques as can be seen in the following example.

In the query shown in Figure 3(a) $R \rhd\lhd S$ and $R \rhd\lhd T$ are (skewed) foreign-key joins. Only a few tuples in *S* and *T* verify predicates $\sigma_{S.a<10}(S)$ and $\sigma_{T.b>20}(T)$ and most tuples in *R* join precisely with these tuples in *S* and *T*. In the absence of SITs, independence is assumed between all predicates and the selectivity of the original query is estimated as the product of individual join and filter selectivity values. This will produce a very small number, clearly a gross underestimate of the selectivity value. In the presence of the two SITs shown in Figure 3, the two maximal equivalent rewritings shown in Figure 3(b) and 3(c) are explored and one of them is selected as the transformed query plan. Each alternative exploits one available SIT and therefore takes into consideration correlations introduced by one of the skewed joins. Thus, the resulting estimations, although not perfect, have considerably better quality than when base-tables statistics are used.

Because the previous example employed view matching techniques as the main engine to guide transformations, no alternative was explored that exploited both SITs simultaneously. This is a fundamental constraint that results from relying exclusively on materialized view matching to enumerate alternatives. Therefore it is desirable to supplement the enumerated alternatives from materialized view matching with additional alternatives that leverage multiple SITs simultaneously. This is accomplished by using conditional selectivity as a formal framework to reason with selectivity values to identify and exploit SITs for cardinality estimation.

Conditional Selectivity

The concept of conditional selectivity allows expression of a given selectivity value in many different but equivalent ways. This description will focus on conjunctive Select Project Join queries, but the methods herein can be extended to handle more general queries.

An arbitrary SPJ query is represented in a canonical form by first forming the Cartesian product of the tables referenced in the query, then applying all predicates (including joins) to the Cartesian product, and projecting out the desired attributes. Thus, an SPJ query is represented as:

$$q = \pi_{a_1, \ldots, a_{na}} (\sigma_{p1} \wedge \ \ldots \ \wedge p_{np}(R_1 \times \ldots \times R_n))$$

where a_i are attributes of $R_1 \times \ldots \times R_n$, and p_i are predicates over $R_1 \times \ldots \times R_n$ (e.g. $R_1.a \leq 25$, or $R_1.x = R_2.y$).

Each set of predicates $\{p_i\}$ that is applied to $R_1 \times \ldots \times R_n$ results in the subset of tuples that simultaneously verify all p_i. Using bag semantics, projections do not change the size of the output, and therefore projections are omitted from consideration when estimating cardinalities. To estimate the size of the output, or its cardinality, the fraction of tuples in $R_1 \times \ldots \times R_n$ that simultaneously verify all predicates p_i (i.e. the selectivity of all p_i) is approximated, and then this fraction is multiplied by $|R_1 \times \ldots \times R_n|$, which can be obtained by simple lookups over the system catalogs. The use of selectivities to obtain cardinalities results in simpler derivations. The classical definition of selectivity is extended as follows:

Let $\mathcal{R} = \{R_1, \ldots, R_n\}$ be a set of tables, and $P = \{p_1, \ldots, p_j\}$, $Q = \{q_1, \ldots, p_k\}$ be sets of predicates over $R^x = R_1 \times \ldots \times R_n$. The selectivity of p with respect to $\sigma_{q1} \wedge \ \ldots \ \wedge_{qk} (R^x)$, denoted $Sel_R(P|Q)$, is defined as the fraction of tuples in $\sigma_{q1} \wedge \ \ldots \ \wedge_{qk} (R^x)$ that simultaneously verify all predicates in P. Therefore,

$$Sel_{\mathcal{R}}(P|Q) = \frac{|\sigma_{p_1} \wedge \ldots \wedge_{p_j} (\sigma_{q_1} \wedge \ldots \wedge_{q_k} (R_1 \times \ldots \times R_n))|}{|\sigma_{q_1} \wedge \ldots \wedge_{q_k} (R_1 \times \ldots \times R_n)|}$$

If Q = Ø, this reduces to $Sel_{R}(P)$, which agrees with the traditional definition of selectivity.

In this description, *tables*(*P*) denotes the set of tables referenced by a set of predicates *P*, and *attr*(*P*) denotes the set of attributes mentioned in *P*. To simplify the notation, "P,Q" denotes "$P \cup Q$" and "p,Q" denotes "$\{p\} \cup Q$", where *p* is a predicate and *P* and *Q* are sets of predicates. For example, given the following query:

```
SELECT * FROM R,S,T
WHERE R.x=S.y AND S.a<10 and T.b>5
```

the selectivity of *q*, $Sel_{\{R,S,T\}}(R.x{=}S.y, S.a{<}10, T.b{>}5)$ is the fraction of tuples in RST that verify all predicates. Additionally, tables($R.x{=}S.y, S.a{<}10$) = $\{R,S\}$, and attr($R.x{=}S.y, S.a{<}10$) = $\{R.x, S.y, S.a\}$.

In general the task is to estimate $Sel_{R}(p1,\ldots,p_k)$ for a given query $\sigma_{p1} \wedge \ldots \wedge_{pk}$ (R^x). Two properties, atomic decomposition and separable decomposition, are verified by conditional selectivity values and allow a given selectivity to be expressed in many equivalent ways. Proofs of the properties are omitted.

Atomic decomposition is based on the notion of conditional probability and unfolds a selectivity value as the product of two related selectivity values:

$$Sel_{R}(P,Q) = Sel_{R}(\mathrm{P}|\mathrm{Q}) \cdot Sel_{R}(\mathrm{Q})$$

The property of atomic decomposition holds for arbitrary sets of predicates and tables, without relying on any assumption, such as independence. By repeatedly applying atomic decompositions over an initial selectivity value *S*, a very large number of alternative rewritings for S can be obtained, which are called decompositions. The number of different decompositions of $Sel_R(p1,\ldots,p_n)$, denoted by $T(n)$, is bounded as follows: $0.5(n+1)! \leq \mathrm{T(n)} \leq 1.5^n n!$ for $\mathrm{n} \leq 1$.

In the presence of exact selectivity information, each possible decomposition of $Sel_R(P)$ results in the same selectivity value (since each decomposition is obtained through a series of equalities). In reality, exact information may not be available. Instead, a set of SITs is maintained and used to approximate selectivity values. In such cases, depending on the available SITs, some decompositions might be more accurate than others. To determine which decompositions are more accurate, a measure of how accurately S can be approximated using the current set of available SITs is assigned to each decomposition S of $Sel_R(P)$. Then approximating $Sel_R(P)$ can be treated as an optimization problem in which the "most accurate" decomposition of $Sel_R(P)$ for the given set of available SITs is sought.

A naïve approach to this problem would explore exhaustively all possible decompositions of $Sel_R(P)$, estimate the accuracy of each decomposition and return the most accurate one. To improve on this approach, the notion of separability is used. Separability is a syntactic property of conditional selectivity values that can substantially reduce the space of decompositions without missing any useful one. It is said that $Sel_R(P)$ is separable (with Q possibly empty) if non-empty sets X_1 and X_2 can be found such that $P \cup Q = X_1 \cup X_2$ and $tables(X_1) \cap tables(X_2) = \emptyset$. In that case, X_1 and X_2 are said to separate $Sel_R(P)$. For example, given $P=\{T.b=5, S.a<10\}$, $\mathrm{Q}=\{R.x=S.y\}$, and $S=$

$Sel_{\{R,S,T\}}(P|Q)$, $X_1 = \{T.b=5\}$ and $X_2 = \{R.x=S.y, S.a<10\}$ separate S. This is because tables(X_1}=$\{T\}$ and tables (X_2) = $\{R,S\}$. If $S.y = T.z$ were added to Q, the resulting selectivity expression is no longer separable.

Intuitively, $Sel_{\mathcal{R}}(P|Q)$ is separable if $\sigma_{P \wedge Q}(R^{x})$ combines some tables by using Cartesian products. It is important to note, however, that even if the original query does not use any Cartesian product, after applying atomic decompositions some of its factors can become separable. The property of separable decomposition, which is applicable where the independence assumption is guaranteed to hold, follows:

Given that $\{P_1,P_2\}$ and $\{Q_1,Q_2\}$ are partitions of P and Q, respectively, and $X_1=P_1 \cup Q_1$ and $X_2=P_2 \cup Q_2$; and $R_1=tables(X_1)$ and $R_2=tables(X_2)$. $Sel_{\mathcal{R}}(P|Q)$ can be separated into $Sel_{\mathcal{R}}(P_1|Q_1) \cdot Sel_{\mathcal{R}}(P_2|Q_2)$. For example, {T.b = 5} and {R.x=S.y, S.a < 10}can be separated into $S = Sel_{\{R,S,T\}}(T.b = 5, S.a < 10 | R.x=S.y)$ which yields $S= Sel_{\{R,S\}}(R.x=S.y, S.a < 10) \cdot Sel_{\{T\}} (T.b = 5)$.

Using the separable decomposition property, it can be assumed that if $\mathcal{H}$ is a statistic that approximates $Sel_{\mathcal{R}}(P|Q)$ and $Sel_{\mathcal{R}}(P|Q)$ is separable as $Sel_{\mathcal{R}1}(P_1|Q_1) \cdot Sel_{\mathcal{R}2}(P_2|Q_2)$ then there are two statistics $\mathcal{H}_1$ and $\mathcal{H}_2$ that approximate $Sel_{\mathcal{R}1}(P_1|Q_1)$ and $Sel_{\mathcal{R}2}(P_2|Q_2)$ such that: 1) $\mathcal{H}_1$ and $\mathcal{H}_2$ combined require at most as much space as $\mathcal{H}$ does, and 2) the approximation using $\mathcal{H}_1$ and $\mathcal{H}_2$ is as accurate as that of $\mathcal{H}$. For example, $Sel_{\{R,S\}}(R.a<10, S.b > 20)$, is separable as $Sel_{\{R\}}(R.a < 10) \cdot Sel_{\{S\}}(S.a > 20)$. In this situation, using two uni-dimensional histograms $H(R.a)$ and $H(S.b)$ to estimate each factor and then multiplying the resulting selectivity values assuming independence (which is correct in this case) will be at least as accurate as using directly a two dimensional histogram $H(R.a,S.b)$ built on $R \times S$. In fact, the independence assumption holds in this case, so the joint distribution over $\{R.a,S.b\}$ can be estimated correctly from uni-dimensional distributions over $R.a$ and $S.b$. For that reason, statistics that directly approximate separable factors of decompositions do not need to be maintained since such statistics can be replaced by more accurate and space-efficient ones. Therefore, all decompositions $S = S_1 \cdot \ldots \cdot S_n$ for which some S_i is separable can be discarded without missing the most accurate decompositions.

The separable decomposition property and the above assumption can substantially reduce the search space, since consideration of large subsets of decompositions can be avoided. However, in many cases the search space is still very large. To make the optimization problem manageable, some restrictions can be imposed on the way the accuracy of decomposition is measured. A dynamic-programming algorithm can then return the most accurate decomposition for a given selectivity value, provided that the function that measures the accuracy of the decompositions is both monotonic and algebraic.

The error of a decomposition, which measures the accuracy of the available set of statistics approximating the decomposition, must verify two properties, monotonicity and algebraic aggregation. Given $\mathcal{S} = Sel_{\mathcal{R}}(p_1,\ldots,p_n)$ is a selectivity value and $\mathcal{S} = \mathcal{S}_1 \cdot \ldots \cdot \mathcal{S}_k$ is a non-separable decomposition of $\mathcal{S}$ such that $\mathcal{S}_i = Sel_{\mathcal{R}}(P_i|Q_i)$. If statistic $\mathcal{H}_i$ is used to approximate $\mathcal{S}_i$, the error$(\mathcal{H}_i,\mathcal{S}_i)$ is the level of accuracy of $\mathcal{H}_i$ approximating $\mathcal{S}_i$. The value error($\mathcal{H}_i,\mathcal{S}_i$) is a positive real number, where smaller values represent better accuracy. The estimated overall error for $\mathcal{S}= \mathcal{S}_1\ldots\mathcal{S}_k$ is given by an aggregate function $E(e_1,\ldots,e_n)$, where $e_i = error(\mathcal{H}_i,\mathcal{S}_i)$.

E is monotonic if every time that $e_i \leq e'_i$ for all i, $E(e_1,\ldots,e_n) \leq E(e'_1,\ldots,e'_n)$. Monotonicity is a reasonable property for aggregate functions representing overall accuracy: if each individual error e'_i is at least as high as error e_i, then the overall $E(e'_1,\ldots,e'_n)$ would be expected to be at lest as high as $E(e_1,\ldots,e_n)$.

F is distributive if there is a function G such that $F(x_1,\ldots,x_n) = G(F(x_1,\ldots,x_n), F(x_{i+1},\ldots,x_n))$. Two examples of distributive aggregates are `max` (with G=`max`) and `count` (with G=`sum`). In general, E is algebraic if there are distributive functions $F_1,\ldots,F_m$ and a function H such that $E(x_1,\ldots,x_n) = H(F_1(x_1,\ldots,x_n),\ldots,F_m(x_1,\ldots,x_n))$. For example `avg` is algebraic with F_1=`sum`, F_2=`count`, and $H(x,y)=x/y$. For simplicity, for an algebraic E, $E_{merge}(E(x_1,\ldots,x_i),\ldots,(E(x_{i+1},\ldots,x_n))$ is defined as $E(x_1,\ldots,x_n)$. Therefore, `avg`$_{\mathrm{merge}}$(`avg`(1,2),`avg`(3,4))=`avg`(1,2,3,4).

Monotonicity imposes the principle of optimality for error values, and allows a dynamic programming strategy to find the optimal decomposition of $Sel_{\mathcal{R}}(P)$. The principle of optimality states that the components of a globally optimal solution are themselves optimal. Therefore the most accurate decomposition of $Sel_{\mathcal{R}}(P)$ can be found by trying all atomic decompositions $Sel_{\mathcal{R}}(P|Q) = Sel_{\mathcal{R}}(P'|Q) \cdot Sel_{\mathcal{R}}(Q)$, recursively obtaining the optimal decomposition of $Sel_{\mathcal{R}}(Q)$, and combining the partial results. In turn, the key property of an algebraic aggregate E is that a small fixed-size vector can summarize sub-aggregations and therefore the amount of information needed to carry over between recursive calls to calculate error values can be bounded.

Building on the decomposition and error principles discussed above, Figure 4 illustrates a recursive algorithm "getSelectivity" designated generally as 400 for obtaining an accurate approximation of a selectivity value. In general, getSelectivity separates a selectivity value into simpler factors and then recursively calls itself to obtain partial selectivity values that are then combined to obtain the requested selectivity value. The algorithm relies on the error function being monotonic and algebraic, and avoids considering decompositions with separable factors. The pruning technique uses the fact that there is always a unique decomposition of $Sel_{\mathcal{R}}(P)$ into non-separable factors of the form $Sel_{\mathcal{R}i}(P_i)$. In other words, given a desired $Sel_{\mathcal{R}}(P)$, and repeatedly applying the separable decomposition property until no single resulting factor is separable, the same non-separable decomposition of $Sel_{\mathcal{R}}(P)$ will result.

In step 410, the algorithm considers an input predicate P over a set of tables R. The algorithm first checks to see if $Sel_{\mathcal{R}}(P)$ has already been stored in a memoization table indicated as 490. If the value is stored, the algorithm returns that value and the process ends. If the value has not yet been stored, the algorithm determines if the input selectivity value predicate $Sel_{\mathcal{R}}(P)$ is separable and if so separates $Sel_{\mathcal{R}}(P)$ into i factors (step 420). For each factor, getSelectivity is recursively called (Step 460) and the optimal decomposition is obtained for each factor. Then, partial results and errors are combined in steps 470 and 475 and returned. Otherwise, $Sel_{\mathcal{R}}(P)$ is not separable and it is passed to steps 430 and 440 where all atomic decompositions $Sel_{\mathcal{R}}(P'|Q) \cdot Sel_{\mathcal{R}}(Q)$ are tried. For each alternative decomposition, $Sel_{\mathcal{R}}(Q)$ is recursively passed to getSelectivity (Step 460). Additionally, in step 450 $Sel_{\mathcal{R}}(P'|Q)$ is approximated using the best available SITs among the set of available statistics 455. If no single statistic is available in step 450 the $error_{P|Q}$ is set to ∞ and another atomic decomposition of the factor is considered. After all atomic decompositions are explored in steps 440 and 450 the most accurate estimation for $Sel_{\mathcal{R}}(P)$ (493) and its associated error is calculated in steps 470 and 475 and returned (and stored in the table 490). As a byproduct of getSelectivity, the most accurate selectivity estimation for every sub-query $\sigma_{P'}(\mathcal{R}^{\times})$ with $P' \subseteq P$ is obtained. It can be shown that getSelectivity(R,P) returns the most accurate approximation of $Sel_{\mathcal{R}}(P)$ for a given definition of error among all non-separable decompositions. A pseudo code implementation of getSelectivity follows:

```
getSelectivity ($\mathcal{R}$:tables, $P$:predicates over $\mathcal{R}^{\times}$)
Returns ($Sel_{\mathcal{R}}(P)$, error$_P$) such that error$_P$ is best among all non-separable decompositions
    if ($Sel_{\mathcal{R}}(P)$) was already calculated)
        ( $Sel_{\mathcal{R}}(P)$), error$_P$) = memoization_table_lookup($P$)
    else if $Sel_{\mathcal{R}}(P)$ is separable
        get the standard decomposition of $Sel_{\mathcal{R}}(P)$:
            $Sel_{\mathcal{R}}(P)= Sel_{\mathcal{R}1}(P_1) \cdot \ldots \cdot Sel_{\mathcal{R}n}(P_n)$
        ($S_{Pi}$, error$_{Pi}$) = getSelectivity($R_i$,$P_i$) (for each $i$=1..$n$)
        $S_p$=$S_{p1}$…$S_{pn}$
        error$_P$ =$E_{merge}$(error$_{P1}$,…, error$_{Pn}$)
```

```
else // $Sel_{\mathcal{R}}(P)$ is non-separable
    $\text{error}_P = \infty$; best$\mathcal{H}$ = NULL
    for each $P' \subseteq P$, $Q = P - P'$
        // check atomic decomposition $Sel_{\mathcal{R}}(P'|Q)) \cdot Sel_{\mathcal{R}}(Q))$
        $(S_Q\ \text{error}_Q)$ = getSelectivity($\mathcal{R}$,$Q$)
        ($\mathcal{H}$, $\text{error}_{P|Q}$) = best statistic (along with the estimated error) to
        approximate $Sel_{\mathcal{R}}(P'|Q))$ .
        if ($E_{merge}(\text{error}_{P'|Q}, \text{error}_Q) \leq \text{error}_P$)
            $S_p = E_{merge}(\text{error}_{P'|Q}, \text{error}_Q)$
            best$\mathcal{H}$ = $\mathcal{H}$
    $S_{p'|Q}$ = estimation of $Sel_{\mathcal{R}}(P'|Q)$ using best$\mathcal{H}$
    $S_p = S_{p'|Q} \cdot S_Q$
memoization_table_insert($P$, $S_p$, $\text{error}_P$)
return ($S_p$, $\text{error}_P$)
```

The worst-case complexity of getSelectivity is $\mathcal{O}(3^n)$, where n is the number of input predicates. In fact, the number of different invocations of getSelectivity is at most 2^n, on for each subset of P. Due to memoization, only the first invocation to getSelectivity for each subset of P actually produces some work (the others are simple lookups). The running time of getSelectivity for k input predicates (not counting recursive calls) is $\mathcal{O}(k^2)$ for separable factors and $\mathcal{O}(2^k)$ for non-separable factors.

Therefore the complexity of getSelectivity is $O(\sum_{k=1}^{n} \binom{n}{k} \cdot 2^k)$, or $\mathcal{O}(3^n)$. In turn, the space complexity of getSelectivity is $\mathcal{O}(2^n)$ to store in the memoization table selectivity and error values for $Sel_{\mathcal{R}}(p)$ with $p \subseteq P$.

The worst-case complexity of getSelectivity, $\mathcal{O}(3^n)$, can be contrasted with the lower bound of possible decompositions of a predicate, $\mathcal{O}((n+1)!)$. Since $(n+1)!/3^n$ is $\Omega(2^n)$, by using monotonic error functions the number of decompositions that are explored is decreased exponentially without missing the most accurate one. If many subsets of P are separable, the complexity of getSelectivity is further reduced, since smaller problems are solved independently. For instance if $Sel_{\mathcal{R}}(P) = Sel_{\mathcal{R}1}(P_1) \cdot Sel_{\mathcal{R}2}(P_2)$, where $|P_1|=k_1$ and $|P_2|=k_2$, the worst case running time of getSelectivity is $\mathcal{O}(3^{k1}+3^{k2})$, which is much smaller than $\mathcal{O}(3^{k1+k2})$.

In step 450, getSelectivity obtains the statistic $\mathcal{H}$ to approximate $Sel_{\mathcal{R}}(P|Q)$ that minimizes error($\mathcal{H}$, $Sel_{\mathcal{R}}(P|Q)$). This procedure consists of 1)obtaining the set of candidate statistics that can approximate $Sel_{\mathcal{R}}(p|Q)$ and 2)selecting from the candidate set the statistic $\mathcal{H}$ that minimizes error($\mathcal{H}$, $Sel_{\mathcal{R}}(p|Q)$).

In general, a statistic $\mathcal{H}$ consists of a set of SITs. For simplicity the notation is modified to represent SITs as follows. Given query expression $q = \sigma_{p1 \wedge \ldots \wedge pk}(\mathcal{R}^x)$, $\text{SIT}_{\mathcal{R}}(a_1,\ldots,a_j|p_1,\ldots,p_k)$ will be used instead of $\text{SIT}(a_1,\ldots,a_j|q)$. That is, the set of predicates of q over $\mathcal{R}^x$ is enumerated, which agrees with the notation for selectivity values. It should be noted that for the purposes of this discussion SITs will be described as histograms, but the general ideas can be applied to other statistical estimators as well. Therefore $\mathcal{H}_{\mathcal{R}}(a_1,\ldots,a_j|p_1,\ldots,p_k)$ is a multidimensional histogram over attributes $a_1,\ldots,a_j$ built on the result of executing $\sigma_{p_1 \wedge \ldots \wedge p_k}(\mathcal{R}^{\times})$. As a special case, if there are no predicates p_i, $\mathcal{H}_{\mathcal{R}}(a_1,\ldots,a_j|)$ is written, which is a traditional base-table histogram. The notion of predicate independence is used to define the set of candidate statistics to consider for approximating a given $Sel_{\mathcal{R}}(P|Q)$.

Given sets of predicates P_1, P_2, and Q it is said that P_1 and P_2 are independent with respect to Q if the following equality holds $Sel_{\mathcal{R}}(P_1,P_2|Q) = Sel_{\mathcal{R}}(P_1|Q) \cdot Sel_{\mathcal{R}}(P_2|Q)$ where R_1=tables(P_1,Q) andR_2 = tables(P_2,Q). If P_1 and P_2 are independent with respect to Q, then $Sel_{\mathcal{R}}(P_1|P_2,Q)= Sel_{\mathcal{R}1}(P_1|Q)$ holds as well. If there is no available statistic approximating $Sel_{\mathcal{R}}(p|Q)$, but there is an available statistic $\mathcal{H}$ approximating $Sel_{\mathcal{R}}(p|Q')$, independence between P and Q' is assumed with respect to Q-Q' and $\mathcal{H}$ is used to approximate $Sel_{\mathcal{R}}(p|Q)$. This idea is used to define a candidate set of statistics to approximate $Sel_{\mathcal{R}}(P|Q)$.

Given that $\mathcal{S}= Sel_{\mathcal{R}}(P|Q)$ where P is a set of filter predicates, such as {R.a<5 S.b>8}, the candidate statistics to approximate $\mathcal{S}$ are all $\{H_{\mathcal{R}}(A|Q'))\}$ that simultaneously verify the following three properties. 1) $attr(P) \subseteq A$ (the SIT can estimate the predicates). 2) $Q' \subseteq Q$ (assuming independence between P and Q-Q'). In a traditional optimizer, $Q'=\varnothing$, so P and Q are always assumed independent. 3) Q' is maximal, i.e., there is no $H_{\mathcal{R}}(A|Q'')$ available such that $Q' \subset Q'' \subseteq Q$.

In principle, the set of candidate statistics can be defined in a more flexible way, e.g., including statistics of the form $H_{\mathcal{R}}(A|Q')$, where Q' subsumes Q. The candidate sets of statistics are restricted as described above to provide a good tradeoff between the efficiency to identify them and the quality of the resulting approximations. For example given $S= Sel_{\mathcal{R}}(R.a<5|p_1,p_2)$ and the statistics $H_R(R.a|\, p_1)$, $H_R(R.a|\, p_2)$, $H_R(R.a|\, p_1,p_2,p_3)$, and $H_R(R.a)$, the set of candidate statistics for S include $\{H_R(R.a|\, p_1)\}$ and $\{H_R(R.a|\, p_2)\}$. $H_R(R.a)$ does not qualify since its query expression is not maximal; and $H_R(R.a|\, p_1,p_2,p_3)$, does not qualify since it contains an extra predicate p_3.

In many cases a predicate P is composed of both filter and join predicates, e.g., $P=\{T.a < 10, R.x=S.y, S.b > 5\}$. To find $Sel_{\mathcal{R}}(P|Q)$ in this case several observations about histograms are used. If $\mathcal{H}_1 = H_R(x,X|Q)$ and $\mathcal{H}_2 = H_R(y,Y|Q)$ and both are SITs, the join $\mathcal{H}_1 \rhd\!\lhd_{x=y} \mathcal{H}_2$ returns not only the value $Sel_{\mathcal{R}}(x=y|Q)$ for the join, but also a new histogram $\mathcal{H}_j = H_R(x,X,Y|x=y,Q)$. Therefore $\mathcal{H}_j$ can be used to estimate the remaining predicates involving attributes $x(=y)$, X, and Y. As an example, to find $Sel_{\mathcal{R}}(R.a < 5, R.x=S.y|Q)$ given histograms $\mathcal{H}_1 = H_{R1}(R.x,R.a|Q)$ and $\mathcal{H}_2 = H_{R2}(S.y|Q)$, the join $\mathcal{H}_1 \rhd\!\lhd_{R.x=S.y} \mathcal{H}_2$ returns the scalar selectivity value $s_1= Sel_{\mathcal{R}}(R.x=S.y\ |Q)$ and also $\mathcal{H}_3 = H_{\mathcal{R}}(R.a|R.x=S.y,Q)$. The selectivity of $Sel_{\mathcal{R}}(R.x=S.y,R.a < 5|Q)$ is then conceptually obtained by the following atomic decomposition: $s_1 \cdot s_2 = = Sel_{\mathcal{R}}(R.a < 5|R.x{=}S.y,Q) \cdot Sel_{\mathcal{R}}(R.x{=}S.y|Q)$, where s_2 is estimated using $\mathcal{H}_3$.

As the example shows, $Sel_{\mathcal{R}}(P|Q)$ can be approximated by getting SITs covering all attributes in P, joining the SITs, and estimating the remaining range predicates in P. In general, the set of candidate statistics to approximate $Sel_{\mathcal{R}}(P|Q)$ is conceptually obtained as follows: 1) All join predicates in P are transformed to pairs of wildcard selection predicates P'. For instance, predicate $R.x{=}S.y$ is replaced by the pair ($R.x$ = ?, $S.y$=?), and therefore $Sel_{\mathcal{R}}(R.x{=}S.y, T.a{<}10, S.b{>}5|Q)$ results in $Sel_{\mathcal{R}}(R.x{=}?, S.y{=}?, T.a{<}10, S.b{>}5|Q)$. 2) Because the join predicates in P were replaced with filter predicates in P' above, the resulting selectivity value becomes separable. Applying the separable decomposition property yields $Sel_{\mathcal{R}1}(P'_1|Q_1) \cdot \ldots \cdot Sel_{\mathcal{R}k}(P'_k|Q_k)$, where no $Sel_{\mathcal{R}i}(P'_i|Q_i)$ is separable. 3) Each $Sel_{\mathcal{R}i}(P'_i|Q_i)$ contains only filter predicates in P'_i, so each candidate set of statistics can be found independently. In order to approximate the original selectivity value with the candidate set of statistics obtained in this way, all $\mathcal{H}_i$ are joined by the attributes mentioned in the wildcard predicates and the actual range of predicates is estimated as in the previous example.

Once the set of candidate statistics is obtained to approximate a given $Sel_{\mathcal{R}}(P|Q)$, the one that is expected to result in the most accurate estimation for $Sel_{\mathcal{R}}(P|Q)$ must be selected, i.e., the statistic $\mathcal{H}$ that minimizes the value of $error(\mathcal{H}, Sel_{\mathcal{R}}(P|Q))$.

In getSelectivity, $error(\mathcal{H},\mathcal{S})$ returns the estimated level of accuracy of approximating selectivity $\mathcal{S}$ using statistic $\mathcal{H}$. There are two requirements for the implementation of $error(\mathcal{H},\mathcal{S})$. First, it must be efficient, since $error(\mathcal{H},\mathcal{S})$ is called in the inner loop of getSelectivity. Very accurate but inefficient error functions are not useful, since the overall optimization time would increase and therefore exploiting SITs becomes a less attractive alternative. For instance, this requirement bans a technique that looks at the actual data tuples to obtain exact error values.

The second requirement concerns the availability of information to calculate error values. At first sight, it is tempting to reformulate error as a meta-estimation technique. Then, in order to estimate the error between two data distributions (actual selectivity values $\mathcal{S}$ vs. SIT-approximated selectivity values $\mathcal{H}$) additional statistics, or meta-statistics could be maintained over the difference of such distributions. Therefore, estimating $error(\mathcal{H},\mathcal{S})$ would be equivalent to approximate range queries over these meta-statistics. However, this approach is flawed, since if such meta-statistics existed, they could be combined with the original statistic to obtain more accurate results in the first place. As an example given $\mathcal{H} = H_R(R.a|p_1)$, approximating S = $Sel_{\mathcal{R}}(R.a<10|p_1,p_2)$. If a meta-statistic $\mathcal{M}$ is available to estimate values error($\mathcal{H}$, $Sel_{\mathcal{R}}(c_1 \leq R.a \leq c_2|p_1,p_2)$), $\mathcal{H}$ and $\mathcal{M}$ can be combined to obtain a new statistic that directly approximates $Sel_{\mathcal{R}}(R.a<10|p_1,p_2)$.

Therefore error values must be estimated using efficient and coarse mechanisms. Existing information such as system catalogs or characteristics of the input query can be used but not additional information created specifically for such purpose.

Application serial number 10/191,822 introduced an error function, *nInd*, that is simple and intuitive, and uses the fact that the independence assumption is the main source of errors during selectivity estimation. The overall error of a decomposition is defined as $\mathcal{S} = Sel_{\mathcal{R}1}(P_1|Q_1) \cdot \ldots \cdot Sel_{\mathcal{R}n}(P_n|Q_n)$ when approximated, respectively, using $\mathcal{H}_{\mathcal{R}1}(\mathrm{A}_1|\mathrm{Q'}_1), \ldots, \mathcal{H}_{\mathcal{R}n}(\mathrm{A}_n|\mathrm{Q'}_n)$ $(Q'_1 \subseteq Q_i)$, as the total number of predicate independence assumptions during the approximation, normalized by the maximum number of independence assumptions in the decomposition (to get a value between 0 and 1). In symbols, this error function is as follows:

$$\mathrm{nInd}\ (\{Sel_{\mathcal{R}}(P_i|Q_i), \mathcal{H}_{\mathcal{R}}(\mathrm{A}_i|\mathrm{Q'}_i)\}) = \frac{\sum_i |P_i| \cdot |Q_i - Q'_i|}{\sum_i |P_i| \cdot |Q_i|}$$

Each term in the numerator represents the fact that *Pi* and Q_i - Q'_i are independent with respect to Q_i, and therefore the number of predicate independent assumptions is $|Pi| \cdot |Q_i - Q'_i|$. In turn, each term in the denominator represents the maximum number of independence assumptions when $Q'_i = \varnothing$, i.e $|P_i| \cdot |Q_i|$. As a very simple example, consider $\mathcal{S} = Sel_R(R.a<10, R.b>50)$ and decomposition $\mathcal{S} = Sel_R(R.a<10|R.b>50) \cdot Sel_R(R.b>50)$. If base table histograms *H*(*R.a*) and *H*(*R.b*) are used, the error using *nInd* is $\frac{1 \cdot (1-0) + 1 \cdot (0-0)}{1 \cdot 1 + 1 \cdot 0} = 1/1 = 1$, i.e., one out of one independence assumptions (between

$R.a < 10$ and $R.b > 50$). *nInd* is clearly a syntactic definition which can be computed very efficiently.

While *nInd* is a very simple metric, often many alternative SITs are given the same score, and *nInd* needs to break ties arbitrarily. This behavior is problematic when there are two or more available SITs to approximate a selectivity value, and while they all result in the same "syntactic" *nInd* score, the actual benefit of using each of them is drastically different, as illustrated in the following example.

Consider $R \bowtie_{R.s=S.s} (\sigma_{S.a<10}S) \bowtie_{S.t=T.t} T$, with both foreign-key joins, and the following factor of a decomposition: $S_1 = Sel_{RST}(S.a<10|R \bowtie S, S \bowtie T)$. If the only candidate SITs to approximate S_1 are $H_1 = H_{\{R,S\}}(S.a|R \bowtie S)$ and $H_2 = H_{\{R,S\}}(S.a|S \bowtie T)$, using the error function *nInd*, each statistic would have a score of 1/2, meaning that in general each alternative will be chosen at random 50% of the time. However, in this particular case H_1 will always be more helpful than H_2. In fact, since $S \bowtie_{S.t=T.t} T$ is a foreign key join, the distribution of attribute $S.a$ over the result of $S \bowtie_{S.t=T.t} T$ is exactly the same as the distribution of $S.a$ over base table S. Therefore, $S \bowtie_{S.t=T.t} T$ is actually independent of $S.a < 10$ and H_2 provides no benefit over the base histogram $H(S.a)$.

An alternative error function, *Diff*, is defined as follows. A single value, $diff_H$, between 0 and 1 is assigned to each available SIT $H = H(R.a|Q)$. In particular, $diff_H = 0$ when the distribution of $R.a$ on the base table R is exactly the same as that on the result of executing query expression Q. On the other hand, $diff_H = 1$ when such distributions are very different (note that in general there are multiple possible distributions for which $diff_H = 1$, but only one for which $diff_H = 0$). Using *diff* values, the *Diff* error function generalizes *nInd* by providing a less syntactic notion of independence. In particular, the overall error value of a decomposition $S = Sel_{\mathcal{R}_1}(P_1|Q_1) \cdot \ldots \cdot Sel_{\mathcal{R}_n}(P_n|Q_n)$ when approximated using $H_1,\ldots,H_n$, respectively is given by:

$$Diff(\{Sel_{\mathcal{R}}(P_i|Q_i),H_i\}) = \frac{\sum_i |P_i| \cdot |Q_i| \cdot |1 - diff_{H_i}|}{\sum_i |P_i| \cdot |Q_i|}$$

The intuition behind the expression above is that the value $|Q_i| \cdot (1\text{-}diff_{Hi})$ in the numerator represents a "semantic" number of independence assumptions when approximating S_i with H_i, and replaces the syntactic value $|\, Q_i - Q'_i \,|$ of *nInd*. In fact in the previous example, $diff_{H1} = 0$, and H_1 effectively contributes the same as a base-table histogram $H(S.a)$, so in that case the error function is 1 (the maximum possible value). In contrast, for $H_2 = H(S.a|R \triangleright\triangleleft S)$, the more different the distributions of $S.a$ on S and on the result of executing $R \triangleright\triangleleft S$, the more likely that H_2 encodes the dependencies between $S.a$ and $\{R \triangleright\triangleleft S, S \triangleright\triangleleft T\}$, and therefore the lower the overall error value.

For $H = H_{\mathcal{R}}(a|Q)$ with $\mathcal{R}'$ denoted as $\sigma_Q(\mathcal{R})$ (the result of evaluating Q over $\mathcal{R}$), $diff_H$ can be defined as:

$$diff_H = 1/2 \cdot \sum_{x \in dom(a)} \left(\frac{f(R,x)}{|R|} - \frac{f(R',x)}{|R'|}\right)^2$$

where $f(\mathcal{R},x)$ is the frequency of value x in $\mathcal{R}$ ($diff_H$ is the squared deviation of frequencies between the base table distribution and the result of executing H's query expression). It can be shown that $0 \leq diff_H \leq 1$, and that $diff_H$ verifies the properties stated above. Values of *diff* are calculated just once and are stored with each histogram, so there is no overhead at runtime. $diff_{H_{R(a|Q)}}$ can be calculated when $H_{R(a|Q)}$ is created, but that might impose a certain overhead to the query processor to get the $f(R,a)$. Instead, $diff_H$ is approximated by carefully manipulating both H and the corresponding base-table histogram (which, if it does not exist, can be efficiently obtained using sampling). The procedure is similar to calculating the join of two histograms.

In essence, *Diff* is a heuristic ranking function and has some natural limitations. For instance, it uses a single number (*Diff*) to summarize the amount of divergence between distributions, and it does not take into account possible cancellation of errors among predicates. However, the additional information used by *Diff* makes it more robust and accurate than *nInd* with almost no overhead.

Referring again to the query of Figure 3, the value of $Sel_{\{R,S,T\}}(\sigma_a, \sigma_b, \bowtie_{RS}, \bowtie_{RT})$ is to be estimated where σ_a and σ_b represent the filter predicates over $S.a$ and $T.b$, respectively, and $\bowtie_{RS}$ and $\bowtie_{RT}$ represent the foreign key join predicates. Using *nInd* for errors, getSelectivity returns the decomposition; $S_1 = Sel_{R,S,T}(\sigma_a|\sigma_b, \bowtie_{RS}, \bowtie_{RT}) \cdot Sel_{R,S,T}(\sigma_b| \bowtie_{RS}, \bowtie_{RT}) \cdot Sel_{R,S,T}(\bowtie_{RS} | \bowtie_{RT}) \cdot Sel_{R,T}(\bowtie_{RT})$ using respectively, statistics $H_{R,S}(S.a|\bowtie_{RS})$ $H_{R,T}(T.b|\bowtie_{RT})$, $\{H_R(R.s), H_S(S.s)\}$, and $\{H_R(R.t), H_T(T.t)\}$. Therefore, both available SITs are exploited simultaneously, producing a much more accurate cardinality estimate for the original query than any alternative produced by previous techniques.

Integration with an Optimizer

The algorithm getSelectivity can be integrated with rule based optimizers. For $q = \sigma_{p1} \wedge \ldots \wedge_{pk}(\mathcal{R}^{x})$, getSelectivity $(\mathcal{R}, \{p_1,\ldots,p_k\})$ returns the most accurate selectivity estimation for both q and all its sub-queries, i.e., $Sel_{\mathcal{R}}(\mathrm{P})$ for all $P \subseteq \{p_1,\ldots,p_k\}$. A simple approach to incorporate getSelectivity into an existing rule-based optimizer is to execute getSelectivity before optimization starts, and then use the resulting memoization table to answer selectivity requests over arbitrary sub-queries. This approach follows a pattern similar to those used by prior art frameworks to enumerate candidate sub-plans, in which a first step generates exhaustively all possible equivalent expressions and then, in a second phase, the actual search and pruning is performed. It was later established that this separation is not useful, since only a fraction of the candidate sub-plans generated during exploration is actually considered during optimization. Instead, newer frameworks interleave an exploration by demand strategy with the search and pruning phase.

Cascades is a state-of-the-art rule-based optimization framework. During the optimization of an input query, a Cascades based optimizer keeps track of many alternative sub-plans that could be used to evaluate the query. Sub-plans are grouped together into equivalence classes, and each equivalence class is stored as a separate node in a memoization table (also called memo). Thus, each node in the memo contains a list of entries representing the logically equivalent alternatives explored so far. Each entry has the form $[op, \{input_1,\ldots,input_n\}, \{parameter_1,\ldots,parameter_k\}]$, where op is a logical operator, such as `join`, $input_i$ is a pointer to some other node (another class of equivalent sub-queries), and $parameter_j$ is a parameter for operator op.

Figure 5 illustrates a memo that corresponds to an intermediate state while optimizing the query SELECT * FROM R, S WHERE R.x=s.Y AND R.a<10 AND S.b>5. The node at the top of the figure groups together all query plans equivalent to $(\sigma_{R.a<10}(R)) \bowtie_{R.x=S.y} (\sigma_{S.b>5}(S))$ that were already explored. The first entry in such node is [SELECT, $\{R \bowtie_{R.x=S.y} (\sigma_{S.b>5}(S))\}$, $\{R.a<10\}$], that is, a filter operator, with parameter $R.a<10$, applied to the node that groups all equivalent expressions for sub-query $R \bowtie_{R.x=S.y} (\sigma_{S.b>5}(S))$. Analogously, the second entry corresponds to a join operator applied to two other nodes.

During optimization, each node in the memo is populated by applying transformation rules to the set of explored alternatives. Rules consist of antecedent and consequent patterns, and optional applicability conditions. The application of a given transformation rule is a complex procedure that involves: (i) finding all bindings in the memo, (ii) evaluating rule preconditions, (iii) firing the rule, i.e., replacing the antecedent pattern with the consequent pattern, and (iv) integrating the resulting expression (if it is new) to the memo table. As a simple example, the first entry in the node at the top of Figure 5 could have been obtained from the second entry by applying the following transformation rule: $[T_i] \bowtie (\sigma_P(T_2)) \Rightarrow \sigma_P (T_1 \bowtie T_2)$ which pulls out selections above join predicates (T_1 and T_2 function as placeholders for arbitrary sub-queries).

The algorithm getSelectivity can be integrated with a Cascades based optimizer. If the optimizer restricts the set of available statistics, e.g., handles only uni-dimensional SITs, then getSelectivity can be implemented more efficiently without missing the most accurate decomposition. For uni-dimensional SITs, it can be shown that no atomic decomposition $Sel_{\mathcal{R}}(P) = Sel_{\mathcal{R}}(P'|Q) \cdot Sel_{\mathcal{R}}(Q)$ with $|P'| > 1$ will have a non-empty candidate set of statistics, and therefore be useful. In this case, line 10 in getSelectivity can be changed to:

```
10    for each P' ⊆ P, Q = P-P' such that |P'| ≤ 1 do
```

without missing any decomposition. Using this optimization, the complexity of getSelectivity is reduced from $\mathcal{O}(3^n)$ to $\mathcal{O}\sum_i \binom{n}{i} \cdot i = \mathcal{O}(n \cdot 2^{n-1})$, and the most accurate selectivity estimations will be returned. As a side note, this is the same reduction in complexity as obtained when linear join trees during optimization as opposed to bushy join trees.

The search space of decompositions can be further pruned so that getSelectivity can be integrated with a cascades based optimizer by coupling its execution with the optimizer's own search strategy. This pruning technique is then guided by the optimizer's own heuristics, and therefore might prevent getSelectivity from finding the most accurate estimation for some selectivity values. However, the advantage is that the overhead imposed to an existing optimizer is very small and the overall increase in quality can be substantial.

As explained, for an input SPJ query $q = \sigma_{p1 \wedge \ldots \wedge pk}(\mathcal{R}^x)$, each node in the memoization table of a Cascades based optimizer groups all alternative representations of a sub-query of q. Therefore the estimated selectivity of the sub-query represented by n, i.e., $Sel_{\mathcal{R}}(P)$ for $P \subseteq \{p_1, \ldots, p_k\}$ can be associated with each node n in the memo. Each entry in n can be associated to a particular decomposition of the sub-query represented by n.

The node at the top of Figure 5, which represents all equivalent representations of $(\sigma_{R.a<10}(\mathrm{R})) \rhd\lhd_{R.x=S.y} (\sigma_{S.b>5}(\mathrm{S}))$. The second entry in such node (the join operator) can be associated with the following decomposition: $Sel_{\{R,S\}}(R.s{=}S.y \mid R.a < 10,\ S.b > 5) \cdot Sel_{\{R,S\}}(R.a < 10,\ S.b > 5)$. The first factor of this decomposition is approximated using available statistics as already explained. In turn, the second factor is separable and can be simplified as $Sel_{\{R\}}(R.a < 10) \cdot Sel_{\{S\}}(S.b > 5)$. The estimated selectivity of each factor of the separable decomposition is obtained by looking in the corresponding memo nodes (the inputs of the join entry being processed). Finally the estimations are multiplied together and then by the first factor of the atomic decomposition $Sel_{\{R,S\}}(R.s{=}S.y \mid R.a < 10,\ S.b > 5)$ to obtain a new estimation for $Sel_{\{R,S\}}(R.s{=}S.y,\ R.a < 10,\ S.b > 5)$.

Each entry $\mathfrak{E}$ in a memo node n divides the set of predicates P that are represented by n into two groups: (i) the parameters of $\mathfrak{E}$, that are denoted $\mathrm{p}_{\mathfrak{E}}$, and (ii) the predicates in the set of inputs to $\mathfrak{E}$, denoted $Q_{\mathfrak{E}} = \mathrm{P}\text{-}\mathrm{p}_{\mathfrak{E}}$. The entry $\mathfrak{E}$ in is then associated with the decomposition $Sel_{\mathcal{R}}(P) = Sel_{\mathcal{R}}(\mathrm{P}_{\mathfrak{E}} \mid \mathrm{Q}_{\mathfrak{E}})\ Sel_{\mathcal{R}}(\mathrm{Q}_{\mathfrak{E}})$ where each $Sel_{\mathcal{R}}(\mathrm{Q}_{\mathfrak{E}})$ is separable into $Sel_{\mathcal{R}1}(\mathrm{Q}^1) \cdot \ldots \cdot Sel_{\mathcal{R}k}(\mathrm{Q}^k{}_{\mathfrak{E}})$, where each $Sel_{\mathcal{R}i}(\mathrm{Q}^i{}_{\mathfrak{E}})$ is associated with the i-th input of $\mathfrak{E}$.

In summary, the set of decompositions is restricted in line 10 of getSelectivity to exactly those induced by the optimization search strategy. Each time we apply a transformation rule that results in a new entry $\mathfrak{E}$ in the node associated with $Sel_{\mathcal{R}}(P)$ to obtain the decomposition $S = Sel_{\mathcal{R}}(\mathrm{P}_{\mathfrak{E}} \mid \mathrm{Q}_{\mathfrak{E}})\ Sel_{\mathcal{R}}(\mathrm{Q}_{\mathfrak{E}})$. If S has the smallest error found so far for the current node $Sel_{\mathcal{R}}(P)$ is updated using the new approximation. Therefore, the overhead imposed to a traditional Cascades based optimizer by incorporating getSelectivity results from getting, for each new entry $\mathcal{E}$, the most accurate statistic that approximates $Sel_{\mathcal{R}}(P_{\mathcal{E}} | Q_{\mathcal{E}})$.

So far in this description all input queries have been conjunctive SPJ queries. Disjunctive SPJ queries can also be handled by the discussed techniques. For that purpose, the identity $\sigma_{p1 \vee p2}(\mathcal{R}^{x}) = \mathcal{R}^{x} - \sigma_{\neg p1 \wedge \neg p2}(\mathcal{R}^{x}))$ is used, and the disjunctive query is translated using a de Morgan transformation to selectivity values as $Sel_{\mathcal{R}}(p_1 \vee p_2) = 1 - Sel_{\mathcal{R}}(\neg p_1, \neg p_2)$. The algorithm then proceeds as before with the equality above used whenever applicable. For example, decomposition $Sel_{\{R,S,T\}}(R.a < 5 \vee (S.b > 10 \wedge T.c{=}5))$ is rewritten as $1 - Sel_{\{R,S,T\}}(R.a \geq 5, (S.b \leq 10 \vee T.c \neq 5))$. The second term is separable and is simplified to $Sel_{\{R\}}(R.a \geq 5) \cdot Sel_{\{S,T\}}(S.b \leq 10 \vee T.c \neq 5))$. The second factor can be transformed again to $1 - Sel_{\{S,T\}}(S.b > 10, T.c{=}5)$ which is again separable, and so on.

The techniques discusses can also be extended to handle SPJ queries with Group-By clauses. In the following query

```
SELECT b1,...,bn
FROM R1,...,Rn
WHERE p1 AND...AND pj
GROUP BY a1,...,ak
```

each b_i is either included in $\{a_1, \ldots, a_k\}$ or is an aggregate over columns of $\mathcal{R}^{x}$. The cardinality of q is equal to the number of groups in the output, i.e., the number of distinct values $(a_1,\ldots,a_k)$ in $\sigma_{p1} \wedge \ldots \wedge_{pk}(\mathcal{R}^{\times})$, and is obtained by multiplying $|\mathcal{R}^{\times}|$ by the selectivity of the query below:

SELECT DISTINCT $\mathrm{a}_1,\ldots,\mathrm{a}_k$

FROM $\mathrm{R}_1,\ldots,\mathrm{R}_n$

WHERE $\mathrm{p}_1,\ldots,\mathrm{p}_j$

Thus, to approximate selectivity values of SPJ queries with Group By clauses, the selectivity values for SPJ queries must be estimated with set semantics, i.e., taking into account duplicate values. The definition of conditional selectivity can be extended to handle distinct values as described next. If P is a set of predicates and A is a set of attributes, *tables*$(P/_A)$ is defined as the set of tables referenced either in A or in P, and *attr*$(P/_A)$ is defined as the attributes either in A or referenced in P. $\mathcal{R} = \{\mathcal{R}_1,\ldots,\mathcal{R}_n\}$ is a set of tables and P and Q are sets of predicates over $\mathcal{R}^{\times} = \mathcal{R}_1 \times \ldots \times \mathcal{R}_n$. A and B are sets of attributes over $\mathcal{R}$ such that *attr*$(P/_A) \subseteq B$. The definition of conditional selectivity is extended as:

$$\mathrm{SelR}\ (P/_A \mid Q/_B) \quad = \quad |\pi_A{}^*(\sigma_P(\pi_B{}^*(\sigma_Q(\mathcal{R}^{\times})))| \,/\, |\pi_B{}^*(\sigma_Q(\mathcal{R}^{\times}))|$$

where $\pi_A{}^*(\mathcal{R})$ is a version of the projection operator that eliminates duplicate values.

The notation of $S = Sel_{\mathcal{R}}(P/_A|Q/_B)$ is simplified, if possible, as follows. If B contains all attributes in $\mathcal{R}$, $/_B$ is omitted from S. Similarly, if $A = B$ then $/_A$ is omitted from $\mathcal{S}$. Finally, if B contains all attributes in $\mathcal{R}$ and $Q = \phi$, the selectivity is rewritten as $\mathcal{S} = Sel_{\mathcal{R}}(P/_{A})$. The value $Sel_{\mathcal{R}}(P/_{A})$ is then the number of distinct A values for tuples in $\sigma_{P}(\mathcal{R}^{x})$ divided by $|\ \mathcal{R}^{x}\ |$. Therefore, for a generic SPJ query with a group-by clause, the quantity $Sel_{\mathcal{R}}\ (p_{1},\ldots,p_{j}/\ _{a1,\ldots,ak})$ is to be estimated.

The atomic decomposition definition can be extended as follows. $\mathcal{R}$ is a set of tables, P is a set of predicates over $\mathcal{R}$, and A is a set of attributes in $\mathcal{R}$. Then:

$$Sel_{\mathcal{R}}(P/_{A}) = Sel_{\mathcal{R}}\ (P_{1}/_{A}|P_{2}/_{B}) \bullet Sel_{\mathcal{R}}(P_{2}/_{B}) \text{ where } P_{1} \text{ and } P_{2} \text{ partition } P \text{ and } attr(\mathrm{P}_{1}/_{A}) \subseteq B.$$

This generalized atomic decomposition can be integrated with a rule-based optimizer that implements coalescing grouping transformations for queries with group-by clauses. Coalescing grouping is an example of push-down transformations, which typically allow the optimizer to perform early aggregation. In general, such transformations increase the space of alternative execution plans that are considered during optimization. The coalescing grouping transformation shown in Figure 6 is associated with the following instance of the atomic decomposition property $Sel_{\mathcal{R}}(\rhd\!\lhd/_{A}) = Sel_{\mathcal{R}}\ (\phi/_{A}|\rhd\!\lhd/_{B}) \bullet Sel_{\mathcal{R}}(\rhd\!\lhd/_{B})$. For the general case, the $\rhd\!\lhd$ in the equality is replaced with the corresponding set of predicates.

For SPJ queries the atomic and separable decompositions can be used alone to cover all transformations in a rule-based optimizer. In general, the situation is more complex for queries with group-by clauses. The separable decomposition property can be extended similarly as for the atomic property. In some cases rule-based transformations require the operators to satisfy some semantic properties such as the invariant grouping transformation shown in Figure 7 that requires that the join predicate be defined over a foreign key of R_1 and the primary key of R_2. In this case, specific decompositions must be introduced that mimic such transformations. Using atomic decomposition it is obtained that $Sel_{\mathcal{R}}(\triangleright\triangleleft/_A) = Sel_{\mathcal{R}}\ (\triangleright\triangleleft\,|\,\phi\ /_A) \bullet Sel_{\mathcal{R}}(\phi\,/_A)$. However, if the invariant group transformation $Sel_{\mathcal{R}}(\triangleright\triangleleft/_A) = Sel_{\mathcal{R}}\ (\triangleright\triangleleft) \bullet Sel_{\mathcal{R}}(\phi\,/_A)$ can be applied as well. For that reason the $Sel_{\mathcal{R}}(\triangleright\triangleleft/_A) = Sel_{\mathcal{R}}\ (\triangleright\triangleleft) \bullet Sel_{\mathcal{R}}(\phi\,/_A)$ is used which can be easily integrated with a rule-based optimizer. This transformation is not valid for arbitrary values of P and A, but instead holds whenever the invariant grouping transformation can be applied.

In the context of SITs as histograms, traditional histogram techniques can be exploited provided that they record not only the frequency but also the number of distinct values per bucket. Referring again to Figure 6, $Sel_{\mathcal{R}}(\triangleright\triangleleft/_A) = Sel_{\mathcal{R}}\ (\phi\,/_A\,|\triangleright\triangleleft/_B)$ $\bullet Sel_{\mathcal{R}}(\triangleright\triangleleft/_B)$. $H_{\mathcal{R}}(\mathrm{A}|\triangleright\triangleleft/_B)$ can be used to approximate $Sel_{\mathcal{R}}(\phi\,/_A|\ \triangleright\triangleleft/_B)$. In general, to approximate $Sel_{\mathcal{R}}(\phi\,/_A|\ Q/_B)$, some candidate SITs of the form $H_{\mathcal{R}}(A|Q'\,/_B)$ where $Q' \subseteq Q$ are used.

SITs can be further extended to handle queries with complex filter conditions as well as queries with having clauses. The following query asks for orders that were shipped no more than 5 days after they were placed. SELECT * FROM `orders` WHERE `ship-date - place-date` < 5. A good cardinality estimation of the query cannot be obtained by just using uni-dimensional base-table histograms over columns ship-date and place-date. The reason is that single-column histograms fail to model the strong correlation that exists between the ship and place dates of any particular order. A multidimensional histogram over both ship-date = place-date might help in this case, but only marginally. In fact, most of the tuples in the two-dimensional space ship-date × place-date will be very close to the diagonal ship-date = place-date because most orders are usually shipped a few days after they are placed. Therefore, most of the tuples in orders will be clustered in very small sub-regions of the rectangular histogram buckets. The uniformity assumption inside buckets would then be largely inaccurate and result in estimations that are much smaller than the actual cardinality values.

The scope of SIT(A|Q) can be extended to obtain a better cardinality estimate for queries with complex filter expressions. Specifically, *A* is allowed to be a column expression over *Q*. A column expression over *Q* is a function that takes as inputs other columns accessible in the SELECT clause of *Q* and returns a scalar value. For instance, a SIT that can be used to accurately estimate the cardinality of the query over is *H* = SIT(diff-date|*Q*) where the generating query *Q* is defined as: SELECT ship-date - place-date as diff-date FROM orders. In fact, each bucket in *H* with range $[x_L \ldots x_R]$ and frequency *f* specifies that *f* orders were shipped between x_L and x_R days after they were placed. Thus, the cardinality of the query above can be estimated accurately with a range query (-∞,..,5] over *H*.

This idea can also be used to specify SITs that help estimating the cardinality of queries with group-by and having clauses. The following query:

```
SELECT A, sum (C)
FROM R
GROUP BY A
HAVING avg(B)<10
```

conceptually groups all tuples in R by their A values, then estimates the average value of B in each group, and finally reports only the groups with an average value smaller than 10. The cardinality of this query can be estimated using H_2 = SIT(avgB|Q_2), where the generating query Q_2 is defined as:

```
SELECT avg(B) as avgB
FROM R
GROUP BY A
```

in this case, H_2 is a histogram in which each bucket with range $[x_L \ldots x_R]$ and frequency f specifies that f groups of tuples from R (grouped by A values) have an average value of B between x_L and x_R. Therefore, the cardinality of the original query above can be estimated with a range query, with range $[-\infty \ldots 10]$, over H_2.

It can be seen from the foregoing description that using conditional selectivity as a framework for manipulating query plans to leverage statistical information on intermediate query results can result in more efficient query plans. Although the present invention has been described with a degree of particularity, it is the intent that the invention include all modifications and alterations from the disclosed design falling within the spirit or scope of the appended claims.